Correction du DS 1

Informatique pour tous, deuxième année

Julien Reichert

Pas de correction pour la question de cours...

Exercice 1

Naïvement, il suffit de tester la taille et d'agir en conséquence. Trier une liste de taille si petite est excessif, en pratique. Vérifier que la taille est entre un et trois n'est pas nécessaire car la spécification exclut les autres tailles.

```
def moyenne(l):
    assert 1 <= len(l) <= 3, "Taille incorrecte"
    if len(l) == 1:
        return l[0]
    if len(l) == 2:
        return .6 * max(l) + .4 * min(l)
    return .6 * max(l) + .1 * min(l) + .3 * (sum(l)-max(l)-min(l)) # Autres formules possibles
```

On peut même retirer les tests sur la taille en la mettant artificiellement à 3.

Exercice 2

Pour retirer toute utilisation d'un `if`, on réécrit même quelques fonctions...

```
def abs(nombre):
    signe = str(nombre)[0] # soit '-' (ord donne 45) soit un caractère chiffre (ord donne 48 à 57)
    return nombre*(ord(signe) > 46) - nombre*(ord(signe) < 46)
# Évite de transformer un entier en flottant avec la racine du carré

def min(l):
    mini = l[0]
    for i in range(1, len(l)):
        mini = (mini + l[i] - abs(mini - l[i])) // 2
    return mini

def max(l):
    return -min([-x for x in l])

def moyennebis(l):
    1 / len(l) + 1 / (4 - min([4, len(l)])) # Erreur si len(l) est 0 ou > 3
    for i in range (3 - len(l)):
        l.append(min(l)) # On peut constater que c'est équivalent
    return .6 * max(l) + .1 * min(l) + .3 * (sum(l) - max(l) - min(l)) # Autres formules possibles
```

Exercice 3

```
def additionne_1(p, n): # une pile et un entier relatif
    if n == 0:
        return p
    symboles = ['-', '+']
    for i in range(abs(n)):
        if est_vide(p) or sommet(p) == symboles[n > 0]:
            empiler(p, symboles[n > 0])
        else:
            _ = depiler(p)

import copy

def additionne_2(p1, p_2): # deux piles
    p2 = copy.copy(p_2) # précaution au cas où p1 et p_2 sont la même variable
    while (not est_vide(p2)):
        signe = depiler(p2)
        if est_vide(p1) or sommet(p1) == signe:
            empiler(p1, signe)
        else:
            _ = depiler(p1)
```

Exercice 4

C'est la combinaison de deux exercices du TP 2 :

```
def multiparenthesage(L):
    parentheses = []
    couples = []
    for i in range(1,len(L)+1):
        if L[-i] < 0:
            parentheses.append((L[i],len(L)-i))
        elif L[-i] > 0:
            try:
                (type,position) = parentheses.pop()
                assert(type == -L[-i])
                couples.append((len(L)-i,position))
            except:
                raise ValueError("Mauvais parenthesage")
    if parentheses == []:
        return list(reversed(couples))
    else:
        raise ValueError("Une parenthese ouvrante au moins est de trop")
```

On notera que la lecture de la liste se fait à l'envers. La raison est qu'alors les couples seront ordonnés par position croissante des parenthèses ouvrantes.

Exercice 5

Le principe est de trouver la récursion comme dans le problème standard des tours de Hanoï. Ici, on veut que les anneaux de 0 à k (c'est-à-dire les $k+1$ plus petits) soient alignés, et c'est vrai pour $k = 0$ de base. Ensuite, pour passer de k à $k+1$, on doit savoir comment envoyer k anneaux de n'importe quelle position à n'importe quelle autre position (c'est le problème classique!), en l'occurrence la position où se site l'anneau $k+1$, ce qui peut être en l'occurrence déjà fait.

D'où le programme :

```
def hanoi_base(n, dep, arr):
    if n == 1:
        print("Envoyer le sommet de la pile %d vers la pile %d."%(dep, arr))
    else:
        hanoi_base(n-1, dep, 6-dep-arr)
        hanoi_base(1, dep, arr)
        hanoi_base(n-1, 6-dep-arr, arr)

def hanoi_en_cours(l1, l2, l3):
    l = [l1, l2, l3]
    n = len(l1) + len(l2) + len(l3)
    positions = [0] * n
    for i in range(3):
        for element in l[i]:
            positions[element] = i+1
    dep = positions[0]
    for k in range(1, n):
        arr = positions[k]
        if dep != arr:
            hanoi_base(k, dep, arr)
            dep = arr
```

Exercice 6

```
def pas_de_deux(n):
    if n < 0:
        return pas_de_deux(-n)
    if n < 3:
        return n != 2
    return n % 3 != 2 and pas_de_deux(n // 3)
```